INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1429

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

ELIMINATION OF REDUNDANCY FROM FUNCTIONS DEFINED BY SCHEMES

Didier CAUCAL

Mai 1991

RR-1429

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

ELIMINATION OF REDUNDANCY FROM FUNCTIONS DEFINED BY SCHEMES

Didier CAUCAL

Publication interne n° 583 - Avril 1991 - 22 pages

Programme 2

Abstract. The infinite tree obtained classically by unfolding the definition of a recursive scheme, contains several identical subtrees. When they are identified, the resulting graph is generated by a deterministic graph grammar, if the scheme is monadic. We show how to extract one such a grammar from the scheme.

ELIMINATION DE LA REDONDANCE DE FONCTIONS DEFINIES PAR SCHEMAS

Résumé. L'arbre infini obtenu de façon usuelle par dépliage des règles d'un schéma récursif, a de nombreux sous-arbres identiques. Lorsqu'ils sont identifiés, on obtient un graphe engendrable à l'aide d'une grammaire déterministe de graphes, si le schéma est monadique. On montre comment extraire une telle grammaire à partir du schéma.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(U.R.A. 227)
UNIVERSITE DE RENNES I I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(UNITE DE RECHERCHE DE RENNES)

ELIMINATION OF REDUNDANCY FROM FUNCTIONS DEFINED BY SCHEMES ([1])

Didier CAUCAL
IRISA , Campus de Beaulieu , F35042 Rennes Cedex , France
email : caucal@irisa.fr

Introduction

A good strategy for evaluating functional programs should (1) avoid computing useless values, i.e. values that are never used, and (2) avoid computing twice the same value. Neither point can be detected statically, but point (2) can be done syntactically by a shared representation of the program. This shared representation will be deduced from the text of the program and has to be finite. The method that we use for this construction is to consider a free interpretation of the recursive scheme obtained classically by unfolding the definitions infinitely. The resulting infinite tree is then contracted as much as possible: identical subtrees, corresponding to the same computation are shared. The result is an infinite graph. It turns out that this infinite graph is generated by a finite system of patterns, which will be the shared representation of the program. To clarify matters, let us introduce the classical example of the Fibonacci function :

$$F(n) = \text{if } n \leq 1 \text{ then } 1 \text{ else } F(n-1) + F(n-2) \text{ endif}.$$

An associated recursive program scheme [Ni 75], [Gu 81], in short RPS, can be the following equation :

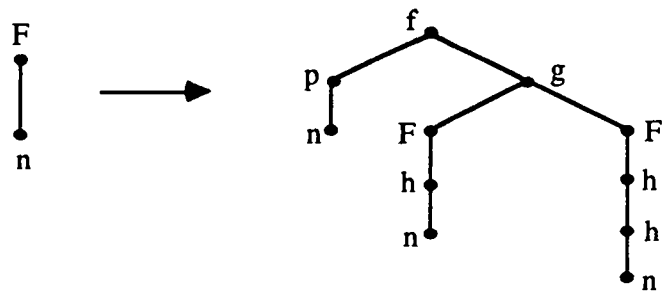
$$F(n) = f(\, p(n) \,,\, g(\, F(h(n)) \,,\, F(h(h(n))) \,) \,)$$

where f,g,p,h are base function symbols with respective arities 2,2,1,1. The base function f is interpreted as the conditional which gives 1 when its condition is realized and its second argument

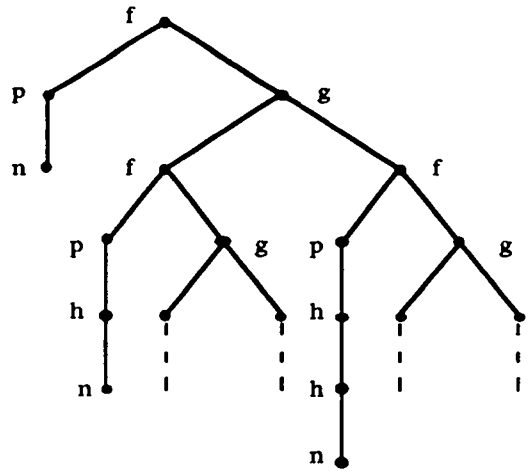
([1]) This work was presented at GRAGRA 90 (GRAph GRAmmar workshop) and will appear in LNCS.

otherwise ; g is the addition ; p is the predicate $n \leq 1$, and h is the decrement function.

Every RPS can be seen as a term rewriting system : the defining equation of F can be oriented to give the following rule :

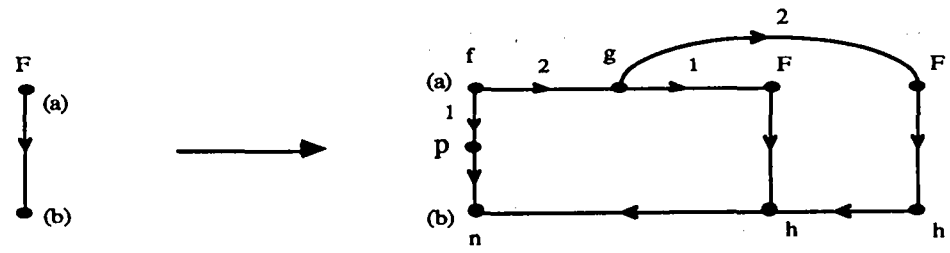


To the defined function F, we associate the infinite tree below, obtained by unfolding the right-hand side of the rule, starting from F(n).

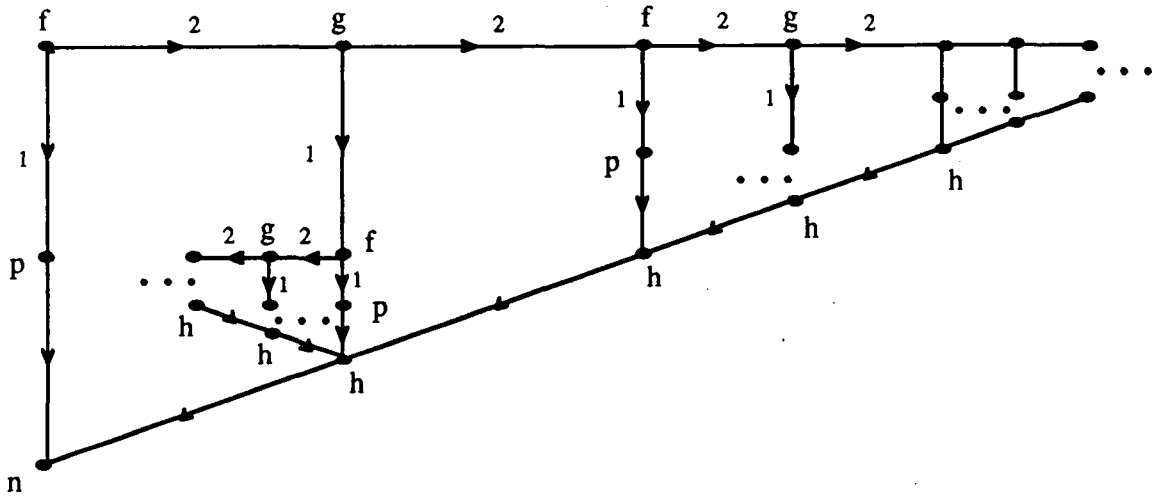


This unfolded tree describes all possible computations of the defined function, but it does not impose any evaluation strategy. Nevertheless, because of the two recursive calls of F in its definition, every evaluation of F by unfolding its tree, is of exponential complexity.

As in [St 80], [Pa 82], [Ba et al. 87], [Ho-Pl 88], we can identify the occurrences of identical subtrees in the definition. We then obtain the following graph grammar where the edge labels indicate the rank of the argument :

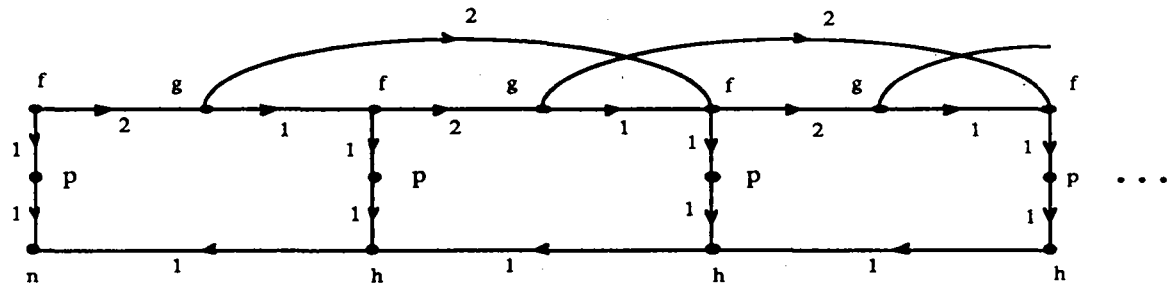


The unfolding of this rule gives the following infinite graph :



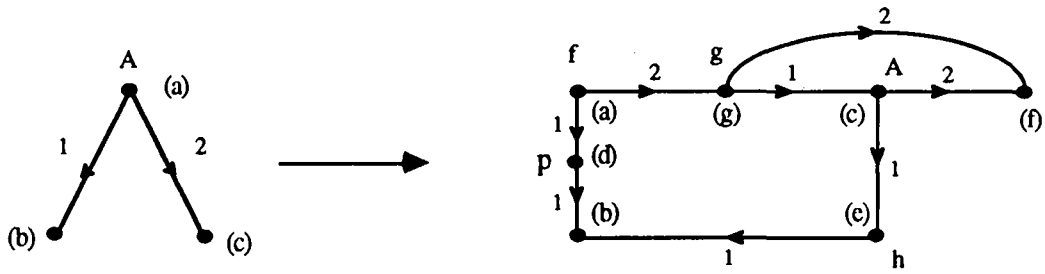
Here again, because of the two recursive calls of F in the right-hand side of the grammar rule, every evaluation of F by unfolding the above graph is of exponential complexity.

There exists a better quotient obtained by identifying all the identical subtrees in the unfolded tree. We obtain the following 'canonical' graph :



Such a graph is a 'pattern' graph, or an equational graph in the sense of Bauderon [Ba 89] and Courcelle [Co 89 a] [Co 89 b], i.e. this graph is obtained by iterated rewritings of a deterministic

(see Section 3) graph grammar, having the following unique rule :



The right-hand side of that rule is the basic pattern of the Fibonacci function. Taking into account the interpretation of the base functions, this grammar can be transformed easily into the following evaluation program of the Fibonacci function :

```
procedure A(a,b,c) ;
   d ← b ≤ 1 ;
   if d then a ← 1 endif ;
   e ← b - 1 ;
   call A(!c,e,!f) ;   {the ! signs mean that the call returns immediately}
   g ← c + f ;                                          {if c and f are evaluated}
   a ← g ;
endprocedure
```

and the execution of A(a,b,c) gives a = Fib(b).

This evaluation program of the Fibonacci function is of complexity linear in time and space.

Of course, the difficulty of this method is to construct the pattern(s) of the canonical graph from the RPS. In this paper, we solve this problem for 'monadic' RPS (meaning that only the base functions may have several arguments). Since there exist [Ca-Mo 90] (or Example 7 in Section 3) canonical graphs of polyadic RPS which are not pattern graphs, this method cannot be applied to all RPS, a fortiori to all term rewriting systems. This is one difference with the approach of [Ho-Pl 87]. Also, our method is by no means akin to an optimized interpreter and in particular the RPS is not required to terminate. It is more like an optimizing compiler, translating the original RPS into a (syntactically) optimal intermediate code : a deterministic graph grammar. One aspect of this optimality consists in getting rid of folding rules during evaluation : this has been taken care of by factoring the infinite tree into the canonical graph.

1. Scheme

This preliminary section recalls basics about recursive program schemes, see among others [Ia 60], [Ga-Lu 73], [Ni 75], [Gu 81], [Co 83].

Definition. A (Greibach) *recursive program scheme* S (in short a scheme), on a graded alphabet F and on a set $X = \{x_1,\ldots,x_n,\ldots\}$ of variables, is a finite set of equations of the form $fx_1\ldots x_n = t$ (where $f \in F_n$, $t \in T(F,\{x_1,\ldots,x_n\})$), satisfying the following conditions :

(i) the scheme is functional : $((fx_1\ldots x_n = s) \in S \wedge (fx_1\ldots x_n = t) \in S) \Rightarrow s = t$

(ii) the scheme is in Greibach form : $(fx_1\ldots x_n = gt_1\ldots t_m) \in S \Rightarrow g \notin \Phi(S)$ where $\Phi(S)$ denotes the set $\{ f \mid (fx_1\ldots x_n = t) \in S \}$ of *defined functions* by S.

Example 1. The recursive function A defined by

$A(n)$ = if $n = 0$ then 0 else $n - A(B(n-1))$ endif

$B(n)$ = if $n = 0$ then 0 else $n - B(A(n-1))$ endif

can be represented by the following scheme :

$A(n) = f(p(n),g(n,A(B(h(n)))))$

$B(n) = f(p(n),g(n,B(A(h(n)))))$.

The equations of a scheme S may be used as rewriting rules, and allow to unfold a given term t into a (usually infinite) derivation tree S(t).

Definition. Given a scheme S and a term $t \in T(F,X)$, the *unfolded tree* S(t) is defined as

(i) $S(t) = f(S(t_1),\ldots,S(t_n))$ if $t = ft_1\ldots t_n$ and $f \notin \Phi(S)$

(ii) $S(t) = S(t'[\, x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n \,])$ if $t = ft_1\ldots t_n$ and $(fx_1\ldots x_n = t') \in S$.

The existence (resp. unicity) of such a tree results from condition (ii) (resp. (i)) of the scheme definition.

Example 2. From the scheme of Example 1, the unfolded tree from An is equal to

$$f(pn,g(n,f(p(f(phn,g(hn,\ldots)),\ldots))) .$$

Unfolded trees generally contain isomorphic subtrees, describing redundant computations. Avoiding redundancy via systematic sharing of isomorphic subtrees leads to 'canonical' graphs. We show in the sequel that canonical graphs turn out to have a regular structure for schemes, satisfying the following three conditions :

(i) the scheme is *monadic* : $\Phi(S) \subseteq F_1$, i.e. every defined function is of arity 1

(ii) the scheme is *without constant* : $F_0 = \emptyset$

(iii) the scheme is *reduced* : $(fx_1 \ldots x_n = t) \in S \Rightarrow S(fx_1 \ldots x_n)$ has a leaf.

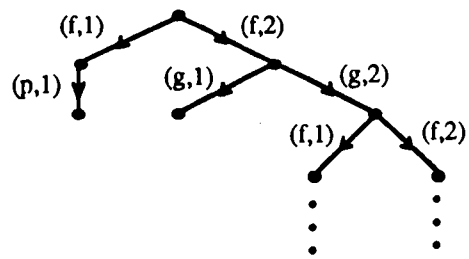
From now on, a scheme is to be understood as a monadic and reduced scheme without constant (we let the reader check that this is the case with Example 1) and the set X of variables is restricted to {x}.

For any graded alphabet F, the 'split' W(F) of F is defined as the set of 'split' symbols :

$$W(F) = \{ (f,i) \mid f \in F_n \wedge 1 \leq i \leq n \} .$$

Given a scheme S, let (fx = t) be an equation in S . All leaves of the unlabelled tree S(fx) are then labeled by x (since S is monadic and uses no constant). In the sequel, we allow ourselves to identify S(fx) with the set of labeled arcs $u \xrightarrow{(g,i)} ui$ where u and ui are nodes of the unfolded tree, and g is the label of u.

Example 3. The tree of Example 2 is identified with the following set of labeled arcs :



To eliminate syntactic redundancy, we identify isomorphic subtrees.

Definition. The *canonical graph* Can(S(fx)) of every unfolded tree S(fx) is defined by

$$\mathrm{Can}(S(fx)) = \{\ [u]_{\equiv} \xrightarrow{a} [v]_{\equiv} \mid (u \xrightarrow{a} v) \in S(fx)\ \}$$

where $u \equiv v$ if and only if the subtrees of S(fx) on u and v are isomorphic.

To decide the isomorphism of subtrees of an unfolded tree T, we associate to every vertex u the *path-language* L(T,u) of the path labels from u to a *terminal vertex* (a vertex which is source of no arc), i.e. L(T,u) is the smallest language over W(F) such that

$$L(T,u) = \bigcup\{\ a.L(T,v) \mid (u \xrightarrow{a} v) \in T\ \}.$$

In particular $L(T,u) = \emptyset$ if u is a terminal vertex. So, the isomorphism of subtrees of T is equivalent to the equality of their path-languages.

Proposition 1.1 *The subtrees of an unfolded tree* S(fx) *on* u *and* v *are isomorphic if and only if* u *and* v *have the same path-languages :* $L(S(fx),u) = L(S(fx),v)$.

Proof.

Clearly, isomorphic subtrees have the same path-language. Conversely, the unfolded tree S(fx) is *deterministic* : two arcs with the same source have different labels. Furthermore S being reduced, from every vertex u of S(fx), there exists a path from u to a terminal vertex. Consequently, the subtree of S(fx) on u is isomorphic to the deterministic graph associated with L(T,u), i.e. the following graph :

$$\{\ v^{-1}.L(T,u) \xrightarrow{a} (va)^{-1}.L(T,u) \mid v \in W(F)^* \wedge a \in W(F) \wedge \exists\, w,\ vaw \in L(T,u)\ \}$$

where $v^{-1}.L(T,u) = \{\ w \mid vw \in L(T,u)\ \}$ is the left quotient of L(T,u) by v.

So, the subtrees of S(fx) are isomorphic if they have the same path-languages. ◆

Now, we need an effective construction of the canonical graph Can(S(fx)) from S and fx.

2. Prefix transition graph

In order to construct the canonical graph Can(S(fx)) of S(fx), we want to show that such a graph is isomorphic to the prefix transition graph [Ca 90] of a reduced simple grammar constructible from S.

Let us recall that a *simple grammar* is an ε-free cf-grammar in Greibach normal form which is LL(1), i.e. for all rules $A \to au$ and $A \to av$ of G where a is a terminal, we have $u = v$. A cf-grammar is *reduced* if G generates a terminal word from any non-terminal.

Given an ε-free cf-grammar in Greibach normal form, and a non-terminal A, the *prefix transition graph* P(G,A) of G accessible from a non-terminal A is the following graph :

$$P(G,A) = \{ u \xrightarrow{a} v \mid A \overset{*}{\underset{G}{\longmapsto}} u \wedge u \overset{a}{\underset{G}{\longmapsto}} v \}$$

where $\overset{a}{\underset{G}{\longmapsto}}$ is a *prefix rewriting step* on the non-terminal words, labelled by a terminal a, and defined by $u \overset{a}{\underset{G}{\longmapsto}} v$ if there exist a rule $X \to ax$ of G and a word y such that $u = Xy$ and $v = xy$,

and $\overset{*}{\underset{G}{\longmapsto}}$ is an arbitrary sequence of such unlabelled steps.

> **Theorem 2.1** *Any pair* (S,f) *of a scheme* S *and an axiom* f *may be effectively transformed into a reduced simple grammar* G *such that the graphs* Can(S(fx)) *and* P(G,f) *are isomorphic.*

The sketch of the proof is given in 3 steps.

Step 1 : we put the scheme S into *Greibach normal form* : each rule has the form $fx = gt_1 \ldots t_n$ where the t_i 's are in $T(\Phi(S),\{x\})$ which corresponds to $\Phi(S)^*.x$ because S is monadic.

To do this, it suffices to replace iteratively each proper tree $t \in (F\text{-}\Phi(S)).(\Phi(S)^*.x)^*$ in a right-hand side of a rule of S by hx where h is a new symbol in F_1, and to add in S the equation $hx = t$. So, we obtain a scheme S' in Greibach form which is equivalent to S : $S'(fx) = S(fx)$ for every $f \in \Phi(S)$.

Example 4. From Example 1, we obtain the following scheme in Greibach normal form :

$Ax = f(Px,Cx)$

$Bx = f(Px,Dx)$

$Cx = g(x,ABHx)$

$Dx = g(x,BAHx)$

$Px = px$

$Hx = hx$.

Step 2 : As [Co-Vu 76], we convert the scheme S in Greibach normal form into a simple grammar G. This elementary transformation amounts to replace each equation $fx = gu_1x...u_nx$ to a grammar rule $f = (g,1)u_1 + ... + (g,n)u_n$ (here, terminals are in the split alphabet $W(F\text{-}\Phi(S))$, and non-terminals are in $\Phi(S)$). So, the language of terminal words generated by G from f is equal to the path-language $L(S(fx),f)$ of the unfolded tree S(fx) (see Proposition 1.1), hence G is reduced.

Example 5. From Example 4, we obtain the following simple grammar

$A = (f,1)P + (f,2)C$

$B = (f,1)P + (f,2)D$

$C = (g,1) + (g,2)ABH$

$D = (g,1) + (g,2)BAH$

$P = (p,1)$

$H = (h,1)$.

Step 3 : Finally, we transform the grammar to an equivalent simple one, and ensures that non-terminal words define always different languages [Co 74]. Such a transformation may be done by a polynomial algorithm [Ca 89]. By Proposition 1.1, the prefix transition graph of the simplified grammar (from the axiom f) is isomorphic to the canonical graph of the tree S(fx).

Example 6. From Example 5, we obtain the following canonical simple grammar :

$A = (f,1)P + (f,2)C$

$C = (g,1) + (g,2)AAH$

$$P = (p,1)$$

$$H = (h,1)$$

which corresponds to the scheme

$$Ax = f(Px,Cx)$$

$$Cx = g(x,AAHx)$$

$$Px = px$$

$$Hx = hx\,.$$

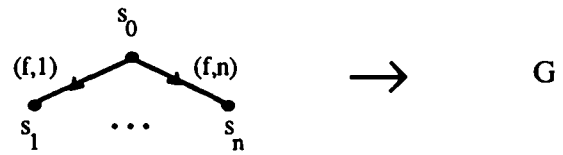
In the next and final section, we show that every prefix transition graph of an ε-free and reduced cf-grammar in Greibach normal form may be effectively generated by finite 'patterns'.

3. Deterministic graph grammar

In this section, we indicate an algorithm which, given a prefix transition graph of a reduced cf-grammar in Greibach normal form, produces its building blocks, called patterns.

Graphs considered henceforth are representation of (possibly infinite) terms : each node s has a finite outdegree, corresponding to some function symbol $f \in F_n$, and s has exactly n outgoing arcs, respectively labeled (f,1) , ... , (f,n). Let us specialize for the above 'term graphs' the usual notion of a graph grammar.

Definition. A *graph grammar* is a finite set of production rules of the form



where $f \in F_n$, $s_i \neq s_j$ if $i \neq j$ and all the s_i are vertices of the finite (term) graph G.

A *non-terminal* of a graph-grammar G is a label of a left-hand side of a rule of G ; a *terminal* of G is a label of a right-hand side of a rule of G which is not a non-terminal.
Graph grammars induce graph rewriting as usual [Ra 84], [Ha-Kr 87], [Ke 89].

Definition. Applying a rule $L \rightarrow R$ to a graph G consists in finding a total mapping g from the set V_R of vertices of R to the set of vertices such that

(i) $g(L) \subseteq G$ where $g(L) = \{ (g(s) \xrightarrow{a} g(t)) \mid (s \xrightarrow{a} t) \in L \}$

(ii) g restricted to $V_R - V_L$ is one-to-one and $g(V_R - V_L) \cap V_G = \emptyset$.

The obtained graph is $(G - g(L)) \cup g(R)$.

The condition (i) amounts to finding the pattern L in G, possibly collapsed. The second condition is an alpha-conversion ensuring that the vertices of R which do not occur in L are also not in G. Beware that the rewriting relation is not a function.

In analogy to Kleene's substitutions on scheme, we introduce parallel rewriting for graphs : G

rewrites in parallel into H if H results from the simultaneous application of the rules at all the occurences of the left members (in G).

We focus now on *deterministic* graph grammars, meaning that there is one rule per non-terminal. Given a graph H represented as a set of arcs and a deterministic graph grammar G, we define $G^{\omega}(H)$ as the unique (up to isomorphism) graph represented by the set of terminal arcs of a sequence $G^n(H)$ where $G^0(H) = H$ and $G^n(H)$ rewrites in parallel into $G^{n+1}(H)$. We use the term *pattern graphs* to denote the class of term graphs $G^{\omega}(H)$ for finite H ; the right members of the rules in G are called the patterns of $G^{\omega}(H)$.

In order to construct the canonical graph Can(S(fx)) of the unfolded tree S(fx) of a scheme S from fx, we want to show that Can(S(fx)) may be effectively generated by a deterministic graph grammar.

Theorem 3.1 *Every canonical graph of a scheme is effectively a pattern graph.*

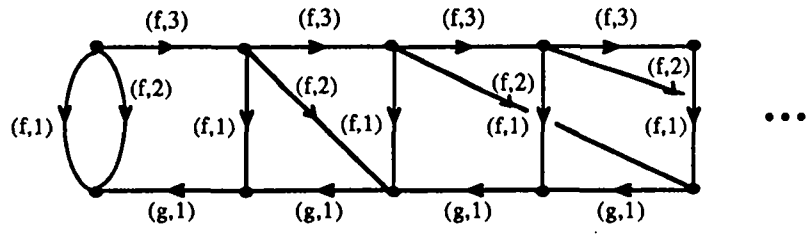
Let us recall that a scheme is to be understood as a reduced monadic scheme without constant. As shown in Example 7 below, Theorem 3.1 is false for polyadic schemes.

Example 7. (given by G. Sénizergues). Let us consider the following reduced and polyadic scheme S without constant :

$$Axy = f(x,y,A(Bx,BBy))$$

$$Bx = gx$$

The canonical graph Can(S(Axx)) of the unlabelled tree of S from Axx is represented below :



and is not a pattern graph.

To prove Theorem 3.1 and from Theorem 2.1, it suffices to establish constructively the result of [Mu-

Sc 85] restricted to reduced simple grammars : every accessible prefix transition graph of a reduced simple grammar is a pattern graph.

Theorem 3.2 *There exist a procedure which from a given reduced ε-free context-free grammar* R *in Greibach normal form, and a letter* r, *produces a deterministic graph-grammar* G *and a finite graph* H *such that the pattern graph generated by* G *from axiom* H *is equal to the prefix transition graph of* R *from* r.

Proof.

The grammar G to be constructed, generates progressively the prefix transition graph

$$P(R,r) = P(R,r)_1 \cup \ldots \cup P(R,r)_n \cup \ldots$$

by slices

$$P(R,r)_n = \{ (s \xrightarrow{a} t) \in P(R,r) \mid |s| = n \}$$

of arcs whose sources are of growing lengths.

We recall [Bü 64] that $\overset{*}{\underset{R}{\longmapsto}} = \underset{R}{\longmapsto}^*$ is decidable and that the language $\{ u \mid r \overset{*}{\underset{R}{\longmapsto}} u \}$ of accessible words by prefix rewriting from a given word r is a rational language recognized by an automaton constructible from (R,r). In particular, the relation

$$\overset{*}{\underset{R}{\longmapsto}} \circ \overset{*}{\underset{R}{\longleftarrow}} = \{ (u,v) \mid \exists w , u \overset{*}{\underset{R}{\longmapsto}} w \wedge v \overset{*}{\underset{R}{\longmapsto}} w \}$$

is decidable.

i) From now on, s is a vertex of P(R,r) different from ε. From s, the grammar G will generate the graph

$$P_s = P(R,r)_s \cup \{ u \xrightarrow{a} v \mid u \in V_{P(R,r)_s} \wedge u \overset{a}{\underset{R}{\longmapsto}} v \wedge |v| < |u| = |s| \}$$

where $P(R,r)_s$ is the connected component of the set

$$\{ (u \xrightarrow{a} v) \in P(R,r) \mid |u| \geq |s| \wedge |v| \geq |s| \} ,$$

containing s.

To establish finiteness of the P_s (up to isomorphism), let us express $P(R,r)_s$ by the rewriting $\overset{*}{\underset{R}{\longmapsto}}_n = (\underset{R}{\longmapsto}_n)^*$ by vertices of length greater or equal to $n \geq 0$, i.e.

$$u \overset{}{\longmapsto}_{R}{}_{n} v \quad \text{if} \quad u \overset{}{\longmapsto}_{R} v \wedge |u| \geq n \wedge |v| \geq n .$$

As $|r| = 1$, we have the following property (1) :

$$P(R,r)_s = \{ u \xrightarrow{a} v \mid \exists t \in V(s) , t \overset{*}{\longmapsto}_{R}{}_{|s|} u \overset{a}{\longmapsto}_{R}{}_{|s|} v \} \qquad (1)$$

where $V(s)$ is the subset of vertices of $P(R,r)_s$ which are targets of arcs in $P(R,r)$ whose sources have of length $< |s|$, and including r if r is a vertex of $P(R,r)_s$, i.e.

$$V(s) = (\{r\} \cup \{ v \mid \exists u , r \overset{*}{\longmapsto}_{R} u \overset{}{\longmapsto}_{R} v \wedge |u| < |s| \}) \cap V_{P(R,r)_s} .$$

We establish in (iii) that all the vertices of $P(R,r)_s$ have a same suffix $Suff(s)$ of length $|s| - 1$ and that P_s is isomorphic to $P(R,V(s).Suff(s)^{-1})$ where

$V(s).Suff(s)^{-1} = \{ u \mid u.Suff(s) \in V(s) \}$ is the right quotient of $V(s)$ by $Suff(s)$

and $P(R,E) = \bigcup \{ P(R,e) \mid e \in E \}$

is the graph of the prefix transitions accessible from a set E of words.

ii) First, we show that $V(s)$ is constructible. As $\overset{*}{\longmapsto}_{R}$ is decidable, we can construct the finite set

$$W(s) = \{r\} \cup \{ v \mid \exists u , r \overset{*}{\longmapsto}_{R} u \overset{}{\longmapsto}_{R} v \wedge |u| < |s| \}$$

of the vertices of $P(R,r)$ accessible by arcs whose sources have of length $< |s|$.

To extract the subset $V(s)$ of $W(s)$, we establish that $V(s)$ is the smallest part of $W(s)$ closed by $\overset{*}{\longmapsto}_{R}{}_{|s|} \circ {}_{|s|}\overset{*}{\longmapsto}_{R}$ and reaching s by $\overset{*}{\longmapsto}_{R}{}_{|s|}$.

Indeed, if $u \overset{*}{\longmapsto}_{R}{}_{|s|} \circ {}_{|s|}\overset{*}{\longmapsto}_{R} v$ with $u \in V(s)$ and $v \in W(s)$ then v is a vertex of $P(R,r)_s$ so $v \in V(s)$. Hence $V(s)$ is a subset of $W(s)$ closed by $\overset{*}{\longmapsto}_{R}{}_{|s|} \circ {}_{|s|}\overset{*}{\longmapsto}_{R}$. Furthermore s is accessible by $\overset{*}{\longmapsto}_{R}{}_{|s|}$ from an element of $V(s)$.

From (1) and for every proper subset P of $V(s)$, there are two paths in $P(R,r)_s$ with the same target, such that the source of one of them is in P and the other is in $V(s) - P$, i.e.

$$\forall P , \emptyset \neq P \subset V(s) , \exists u \in P , \exists v \in V(s) - P , u \overset{*}{\longmapsto}_{R}{}_{|s|} \circ {}_{|s|}\overset{*}{\longmapsto}_{R} v .$$

So, $V(s)$ is included in the smallest part of $W(s)$ containing an element t of $V(s)$ and closed by

$\overset{*}{\underset{R}{\longmapsto}}_{|s|} \circ {}_{|s|}\overset{*}{\underset{R}{\longmapsto}}$. Furthermore, if $t \in W(s)$ and $t \overset{*}{\underset{R}{\longmapsto}}_{|s|} s$ then $t \in V(s)$.

Finally, V(s) is the smallest part of W(s) closed by $\overset{*}{\underset{R}{\longmapsto}}_{|s|} \circ {}_{|s|}\overset{*}{\underset{R}{\longmapsto}}$ whose an element is accessible by ${}_{|s|}\overset{*}{\underset{R}{\longmapsto}}$ from s.

To construct V(s), it remains to decide on $\overset{*}{\underset{R}{\longmapsto}}_{|s|} \circ {}_{|s|}\overset{*}{\underset{R}{\longmapsto}}$ restricted to W(s).

But for every word $u = u'u''$ and $v = v'v''$ of length $\geq |s|$ where $|u''| = |s| - 1 = |v''|$, we have

$$u \overset{*}{\underset{R}{\longmapsto}}_{|s|} \circ {}_{|s|}\overset{*}{\underset{R}{\longmapsto}} v \quad \text{iff} \quad u'' = v'' \wedge \exists\, w \neq \varepsilon,\ u' \overset{*}{\underset{R}{\longmapsto}} w \overset{*}{\underset{R}{\longmapsto}} v' .$$

Then $\overset{*}{\underset{R}{\longmapsto}}_{|s|} \circ {}_{|s|}\overset{*}{\underset{R}{\longmapsto}}$ is decidable and V(s) is constructible.

iii) From (ii), the equivalence relation $\equiv$ on the set of the vertices of P(R,r) different from ε, defined by

$$s \equiv t \quad \text{iff} \quad V(s).Suff(s)^{-1} = V(t).Suff(t)^{-1}$$

is decidable.

Furthermore $\equiv$ is of finite index. Indeed from (ii), Suff(s) is a common suffix to all elements of V(s), and every element of V(s) is of length at most $|s| + K - 1$ where K is the maximal length of the right-hand side of the rules of R.

Finally $\equiv$ is finer than the equivalence relation of the couples (s,t) such that P_s is isomorphic to P_t. Indeed, from (1) and as Suff(s) is a common suffix to all elements of V(s), Suff(s) is a common suffix to all vertices of $P(R,r)_s$, then

$$P(R,r)_s = \{\, u \xrightarrow{a} v \mid \exists\, t \in V(s).Suff(s)^{-1},\ t \overset{*}{\underset{R}{\longmapsto}}_1 u \overset{a}{\underset{R}{\longmapsto}}_1 v \,\}.Suff(s) ,$$

so $P_s = P(R,V(s).Suff(s)^{-1}).Suff(s)$.

iv) We can now construct a graph grammar G generating P(R,r) by vertices of growing length.

As $\equiv$ is an equivalence of finite index, a set A of representatives is constructible. It suffices to construct A by vertices of growing length. In fact, if $s \equiv t$ then $P(R,r)_t = (P(R,r)_s.Suff(s)^{-1}).Suff(t)$ so if $s \overset{a}{\underset{R}{\longmapsto}} s'$ and $|s'| \geq |s|$ then $t \overset{a}{\underset{R}{\longmapsto}} t'$ and $s' \equiv t'$ for $t' = (s'.Suff(s)^{-1}).Suff(t)$.

Take an injection j of A into the subset of the functions of F which are not in R, and such that j(s)

is of arity #V(s) for all $s \in A$. Then to every s in A, we associate a following elementary graph

$$L(s) = \{ \mathrm{Suff}(s) \xrightarrow{(f,i)} s_i \mid 1 \le i \le p \} \text{ where } \{s_1,\ldots,s_p\} = V(s) \text{ and } f = j(s).$$

Such a graph L(s) will be the left-hand side of a rule of G where the right-hand side has as terminal arcs set, the graph

$$T(s) = \{ (u \xrightarrow{a} v) \in P(R,r)_s \mid |u| = |s| \}$$

$$= \{ u \xrightarrow{a} v \mid u \underset{R}{\overset{a}{\longmapsto}} v \wedge |u| = |s| \wedge \exists t \in V(s),\ t \underset{R}{\overset{*}{\longmapsto}}_{|s|} u \}$$

of the arcs of $P(R,r)_s$ of sources of length |s|, and as non-terminal arcs set the graph N(s) to be defined. For this and as R is reduced, to every vertex s of P(R,r) different from ε, we take a minimal set R_s of vertices of $P(R,r)_s$ of length |s| + 1, such that

$$\cup\{ V(t) \mid t \in R_s \} = \cup\{ V(t) \mid t \in V_{P(R,r)_s} \wedge |t| = |s| + 1 \}$$

$$= \{ v \mid \exists u,\ r \underset{R}{\overset{*}{\longmapsto}} u \underset{R}{\longmapsto} v \wedge |u| < |s|+1 \le |v| \}$$

$$= \{ u \in V_{T(s)} \cup V(s) \mid |u| > |s| \}.$$

This allows us to construct the deterministic graph grammar G :

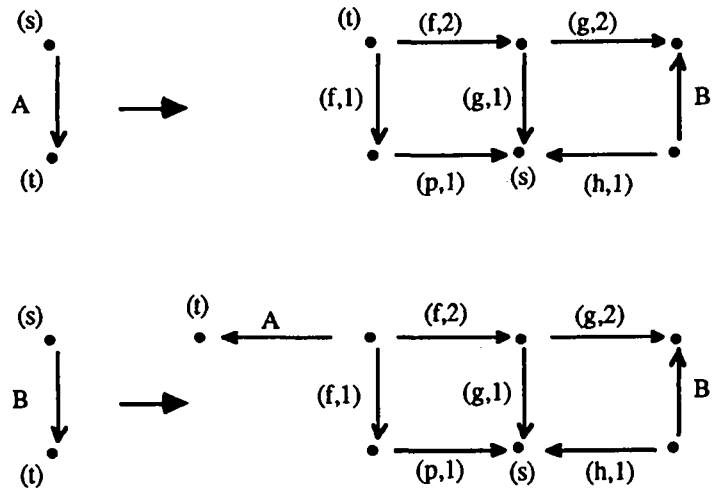
$$G = \{ (L(s), T(s) \cup N(s)) \mid s \in A \}$$

where $N(s) = \{ (L(u).\mathrm{Suff}(u)^{-1}).\mathrm{Suff}(t) \mid t \in R_s \wedge u \in A \wedge u \equiv t \}$.

By construction and for every $s \in A$, $G^{\omega}(L(s))$ is isomorphic to P_s. In particular $G^{\omega}(L(r_0))$ is isomorphic to P_{r_0} where $r_0 \in A$ and $r_0 \equiv r$; so $G^{\omega}(L(r_0))$ is isomorphic to $P_r = P(R,r)$. ◆

A similar result has been given by Baeten, Bergstra and Klop [Ba-Be-Kl 87]. But, our construction is slightly more complex, and is different of [Ca 90], because in our case, the progressive decomposition of the transition graph proceeds by layers of term-graphs, not by layers of nodes.

Example 8. From Example 6, we obtain the following graph grammar :



which generates from A the canonical graph of the unfolded tree of Example 3 (or of the scheme of Example 1).

Even taking into account a such optimization, the evaluation complexity of a recursive function depends on the structure of its definition.

Example 9. From Example 8, we obtain a program of exponential complexity to evaluate the recursive function A of Example 1. But this function can also be defined as follows :

$$A(n) = B(n,0,i(T))$$

$$B(n,m,T) = \text{if } m = n \text{ then } T(m) \text{ else } B(n,m+1,j(m+1,T)) \text{ endif}$$

$$i(T) = T \text{ where } T(0) \leftarrow 0$$

$$j(m,T) = T \text{ where } T(m) \leftarrow m - T(T(m-1))$$

by using a table T. The complexity becomes linear.

This points out that our construction need be carried over also for a class of polyadic schemes.

Conclusion

Given a reduced monadic scheme without constant, we have presented a method to construct a deterministic graph grammar generating the canonical graph of the scheme. This construction has just been extended to a general subclass of polyadic schemes which includes the class of reduced monadic schemes.

Aknowledgments

Let me thank P. Darondeau, R. Monfort and J.-C. Raoult for their help in the drafting of this paper. I thank also a referee for his detailed remarks.

References

Ba-Be Kl 87 J.C.M. Baeten, J.A. Bergstra, J.W. Klop *Decidability of bisimulation equivalence for processes generating context-free languages*, LNCS 259, p. 94-111, 1987.

Ba-Ee-Gl Ke-Pl-Sl 87 H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plesmeijer, M.R. Sleep *Term graph rewriting*, LNCS 259, p. 141-158, 1987.

Ba 89 M. Bauderon *On systems of equations defining infinite graphs*, LNCS 344, p. 54-73, 1989.

Bü 64 R. Büchi *Regular canonical systems*, Archiv für Mathematische Logik und Grundlagenforschung 6, p. 91-111, 1964.

Ca 89 D. Caucal *A fast algorithm to decide on simple grammars equivalence*, LNCS 401, p. 66-85, 1989.

Ca 90 D. Caucal *On the regular structure of prefix rewritings*, LNCS 431, p. 87-102, 1990.

Ca-Mo 90 D. Caucal, R. Monfort *On the transition graphs of automata and grammars*, WG 90, to appear in LNCS, 1990.

Co 74 B. Courcelle *Une forme canonique pour les grammaires simples déterministes*, Rairo 1, p. 19-36, 1974.

Co 83 B. Courcelle *Fundamental properties of infinite trees*, TCS 25, p. 95-169, 1983.

Co 89 a B. Courcelle *The monadic second-order logic of graphs, II : infinite graphs of bounded width*, Math. Syst. Theory 21, p. 187-222, 1989.

Co 89 b B. Courcelle *The definability of equational graphs in monadic second order logic*, LNCS 372, p. 207-221, 1989.

Co-Vu 76 B. Courcelle, J. Vuillemin *Completeness result for the equivalence of recursive schemes*, JCSS 12, p. 179-197, 1976.

Ga-Lu 73 S. Garland, D. Luckam *Program schemes, recursion schemes, and formal languages*, JACM 7, p. 119-160, 1973.

Gu 81 I. Guessarian *Algebraic semantics*, LNCS 79, 1981.

Ha-Kr 87 A. Habel, H.J. Kreowski *Some structural aspects of hypergraph languages generated by hyperedge replacement*, LNCS 247, p. 207-219, 1987.

Ho-Pl 88 B. Hoffmann, D. Plump *Jungle evaluation for efficient term rewriting*, LNCS 343, p. 191-203, 1988.

Ia 60 Ianov *The logical schemes of algorithms*, Problems of cybernetic, USSR, p. 82-140, 1960.

Ke 88 R. Kennaway *On 'on graph rewritings'*, TCS 52, p. 37-58, 1988.

Mu-Sc 85 D. Muller, P. Schupp *The theory of ends, pushdown automata, and second order logic*, TCS 37, p. 51-75, 1985.

Ni 75 M. Nivat *On the interpretation of polyadic recursive schemes*, Symposia Mathematica 15, Academic Press, 1975.

Pa 82 P. Padawitz *Graph grammars and operational semantics*, TCS 19, p. 117-141, 1982.

Ra 84 J.-C. Raoult *On graph rewritings*, TCS 32, p. 1-24, 1984.

St 80 J. Staples *Computation on graph-like expressions*, TCS 10, p. 171-185, 1980.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA 1991

PI 570 **DESIGN DECISION FOR THE FTM : A GENERAL PURPOSE FAUTL TOLERANT MACHINE**
Michel BANATRE, Gilles MULLER, Bruno ROCHAT, Patrick SANCHEZ
Janvier 1991, 30 pages

PI 571 **ANIMATION CONTROLEE PAR LA DYNAMIQUE**
Georges DUMONT, Parie-Paule GASCUEL, Anne VERROUST
Février 1991, 84 pages

PI 572 **MULTIGRID MOTION ESTIMATION ON PYRAMIDAL REPRESENTATIONS FOR IMAGE SEQUENCE CODING**
Nadia BAAZIZ, Claude LABIT
Février 1991, 48 pages

PI 573 **A SURVEY OF TREE-TRANDUCTIONS**
Jean-Claude RAOULT
Février 1991, 18 pages

PI 574 **THE OPTIMAL ADAPTIVE CONTROL USING RECURSIVE IDENTIFICATION**
Anatolij B. JUDITSKY
Février 1991 - 26 pages

PI 575 **MANUEL SIGNAL**
Patricia BOURNAI, Bruno CHERON, Bernard HOUSSAIS, Paul LE
Paul LE GUERNIC
Février 1991, 84 pages

PI 576 **AN INFORMATION BASED RELIABILITY PREDICTOR FOR SYSTEMS IN OPERATIONAL PHASE**
Kamel SISMAIL
Février 1991 - 22 pages

PI 577 **MULTISCALE STATISTICAL SIGNAL PROCESSING AND RANDOM FIELDS ON HOMOGENEOUS TREES**
Albert BENVENISTE, Michèle BASSEVILLE, Ramine NIKOUKHAH,
Alan S. WILLSKY, Ken C. CHOU
Mars 1991 - 18 pages

PI 578 **TOWARDS A DECLARATIVE METHOD FOR 3D SCENE SKETCH MODELING**
Stéphane DONIKIAN, Gérard HEGRON
Mars 1991 - 22 pages

PI 579 **SYSTEMES MARKOVIENS DISCRETS STATIONNAIRES ET APPLICATIONS**
Jean PELLAUMAIL
Mars 1991 - 284 pages

PI 580 **DESCRIPTION ET SIMULATION D'UN SYSTEME DE CONTROLE DE PASSAGE A NIVEAU EN SIGNAL**
Bruno DUTERTRE, Paul LE GUERNIC
Mars 1991 - 66 pages

PI 581 **THE SYNCHRONOUS APPROACH TO REACTIVE AND REAL-TIME SYSTEMS**
Albert BENVENISTE
Avril 1991 - 36 pages

PI 582 **PROGRAMMING REAL TIME APPLICATIONS WITH SIGNAL**
Paul LE GUERNIC, Thierry GAUTIER, Michel LE BORGNE, Claude LE MAIRE
Avril 1991 - 36 pages

PI 583 **ELIMINATION OF REDUNDANCY FROM FUNCTIONS DEFINED BY SCHEMES**
Didier CAUCAL
Avril 1991 - 22 pages

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399